

Aaron Kobayashi
9/25/2002
Branch Prediction

Assumptions:

The basic assumptions of this paper are that the predictors can actually be implemented, and that the simulator yields accurate and precise data.

Summary:

Pipelining is an important component of any modern microprocessors. As the industry continues to push the limits of microprocessor technology, we have seen pipelines become increasingly important. When a branch misprediction occurs, there is a heavy penalty to be paid in the pipelined architectures of today. This penalty continues to become more severe as the pipeline depth increases.

To understand branch predictors, a fundamental understanding of branches is necessary. There are two basic types of branches—conditional and unconditional. Conditional branches are determined at runtime and are based on some condition such as the value of a memory location or a bit in the status register. Unconditional branches can be further divided into immediate branches, indirect branches, and returns. Immediate branches have a target address hard coded at compile time, indirect branches get their target from a register, and returns usually get their target from a stack. Research shows that the majority of branches (72%) are conditional. Research also confirms the rule of thumb that 10% of the code takes 90% of the execution time. These findings coupled with the increasing depth of today's pipelines demonstrate the importance of designing good conditional branch predictors.

Once a basic understanding of the types of branches is obtained, it is important to understand how branches relate to each other. Branches often behave in systematic fashions. The most common behavior is a sequence of a number of taken branches, followed by a not taken one. This sequence would relate to a loop structure where the not taken branch causes the loop to iterate. Branches that are close together often show correlation. This is because they are often based on the same or related variables. Research shows that more than 60% of all branches show some correlation to the outcomes of recently executed branches.

In order to understand where we are going in the field of branch prediction, it is important to understand what technology is currently in use. Basic branch predictors, two-level adaptive predictors, interference reduction predictors, and hybrid predictors are all good examples of the current state of branch prediction. Basic predictors are simple to design and are mainly concerned with predicting the most frequently taken direction of a branch. These predictors have little or no memory, and make assumptions about the structure of the code being executed. While simple in design, these predictors have been shown to have misprediction rates as low as 10%. Two-level adaptive predictors are more complex, but yield higher accuracies by utilizing a second level of history. The two-level predictors collect the outcomes of a series of branches in the first level history, and then use the second level history to predict the outcome of the branch from the last time that pattern was seen. This scheme takes advantage of the fact that branches tend to show correlation to other nearby branches. These predictors have been shown to have misprediction rates as low as 5%. While these predictors represent a great improvement over the basic predictors, they take time to “warm up”, and are much more complex than

the simple predictor design. It is interesting to note that for a modestly sized two-level predictor, as much as one in three mispredictions can be due to interference. An interference reduction predictor attempts to reduce the impact of this interference by not only taking into account the branch stream, but also the addresses of the branches involved. This reduces the amount of interference caused by two branch sequences that have identical taken paths. As we have seen, each predictor type has benefits and drawbacks. The fourth type of predictor, the hybrid predictor, attempts to draw on the benefits of each type by combining two or more predictors into a single more accurate predictor. The hybrid predictor combines the other types of predictors, and adds a selector that chooses which predictor is more likely correct for each branch.

The actual design of a branch predictor takes some further examination. There are three different things that happen to allow a branch prediction—determining that a branch is being fetched, predicting the direction of the branch, and predicting the target of the branch. In the theoretical world, these steps could all happen instantaneously, but in design, they will require multiple cycles to be achieved. The final section of the paper introduces a few techniques for dealing with this limitation. Allowing a predictor to take multiple cycles opens the door for more complex predictors, but we pay the price in complexity of our system. When using a pipelined predictor, history about the not taken branches in the last cycle isn't available unless special support structures are implemented to allow access to this data earlier. Bubbles in the pipeline can seriously lower the utilization of the processor. An instruction buffer will alleviate this under utilization, but increases the complexity of the processor and adds latency to the pipeline. Another way to raise the utilization of the processor is to use a simple single cycle predictor with a multi-cycle override. To avoid bubbles created by taken branches in a static predictor, we can use a simple predictor that can be overridden when the larger predictor makes a prediction. This approach is just like the hybrid processor described above. A further extension of this idea would be to allow the predictor to make predictions two fetches ahead. This will avoid the penalties (bubbles) induced by the simple predictor at the cost of a large increase in complexity.

Limitations:

The paper did not discuss how the predictors would be modified to be suitable for a RISC vs. a CISC environment. Also the use of compiler support was not discussed in detail.

Experimental Results:

This paper was highly theoretical and did not have a great deal of experimental results. Tests were run to evaluate the misprediction rates of the different predictors, showing that the two level predictors yielded much lower misprediction rates than the basic predictors. It was also shown through thorough experimentation that the rule of thumb that 10% of the code is reasonable for 90% of the execution time was valid.

References:

- M. Evers, and T.-Y. Yeh, "[Understanding Branches and Designing Branch Predictors for High Performance Microprocessors.](#)" to appear in Proceedings of IEEE.